



Getting started with the D0 (analysis) software

Marco Verzocchi
University of Maryland

Part 3

Contents

part 1 (8:15-9:00)

- D0 software black magic explained (those setup commands)
- Software versioning and CVS
- Understanding the directory structure of different packages
- How to build an executable
- The D0 software framework
- Framework RCP

part 2 (9:05-9:50)

- More on run control parameters (RCP)
- Understanding the RCP databases, problems with RCP files
- How to share data between packages (the EDM)
- What is the purpose of all those interfaces
- Event filtering
- Input/output
- Do and don't with the gmake command
- Using the d0tools to run D0 programs

Contents

part 3 (10:20-11:05)

- ◆ The D0ChunkAnalyze example
- ◆ Accessing some D0 physics objects from the chunks
- ◆ Writing "Elvis has just left the building" and "Elvis is dead"
- ◆ Filling histograms and ROOT tuples

part 4 (11:10-11:50)

- ◆ Other chunks
- ◆ Chunk documentation
- ◆ Trigger selection
- ◆ Stuff I wanted to cover, but didn't find time for it:
 - ◆ RTE
 - ◆ d0cuts
 - ◆ luminosity calculation and bookkeeping
 - ◆ np_tmb_stream
- ◆ Documentation
- ◆ Yes you can contribute.....

analysis_example/analysis_example/D0ChunkAnalyze.hpp

```
1 #ifndef INC_D0CHUNKANALYZE
2 #define INC_D0CHUNKANALYZE
3 ///////////////////////////////////////////////////////////////////
4 //
5 //
6 // File: D0ChunkAnalyze.hpp
7 //
8 // Purpose: perform simple analysis at the D0 physics objects level
9 //
10 // Created: 07/24/2002 Marco Verzocchi
11 //
12 // History:
13 //
14 //
15 ///////////////////////////////////////////////////////////////////
16
17 // Dependencies (#includes)
18 #include "edm/Event.hpp"
19 #include "edm/TKey.hpp"
20 #include "edm/Tag.hpp"
21
22 #include "framework/Package.hpp"
23 #include "framework/hooks/Process.hpp"
24 #include "framework/hooks/Analyze.hpp"
25 #include "framework/hooks/JobSummary.hpp"
26
27 #include "ErrorLogger/ErrorLog.h"
```

D0ChunkAnalyze.hpp Lines 1-29/160 15%

EDM header files: what is an
Event, how to access Chunks,
event tagging

How to interact with the
framework (interfaces)

Error Logger (discussed in few
slides)

Handle output ROOT files

```
30 #include "HepTuple/HepRootFileManager.h"
31
32 #include "em_evt/EMparticleChunk.hpp"
33 #include "jet_evt/JetChunkSelector.hpp"
34
35 #include "analysis_utilities/D0TriggerSelector.hpp"
36
37 #include <vector>
38
39 #include "TH1.h"
40 #include "TH2.h"
41 #include "TTree.h"
42
43 ///////////////////////////////////////////////////
44
45 namespace D0example {
46
47     class D0ChunkAnalyze :
48     {
49     public fwk::Package, public fwk::Process,
49     public fwk::Analyze, public fwk::JobSummary,
49     public fwk::FileOpen, public fwk::FileClose {
```

Put your classes in
a namespace to avoid
conflicts, particularly
if you all copy from
this example

Interfaces implemented by the
D0ChunkAnalyze package

Already seen in part 2.....

```
50 public :
51
52 // Constructor/Destructor
53 D0ChunkAnalyze(fwk::Context* context);
54 ~D0ChunkAnalyze();
55
56 // Overridden hook methods
57 fwk::Result processEvent(edm::Event &event);
58 fwk::Result analyzeEvent(const edm::Event &event);
59 fwk::Result fileOpen(const fwk::FileInfo &fileinfo);
60 fwk::Result fileClose(const fwk::FileInfo &fileinfo);
61 fwk::Result jobSummary();
62
63 // Overridden package methods
64 std::string packageName() const {return package_name();}
65 static const std::string package_name() {return "D0ChunkAnalyze";}
66 static const std::string version() {return "$Id: D0ChunkAnalyze.hpp,v 1.2
67 2002/08/16 16:16:45 mverzocc Exp $";}
68
```

Return package name

Everything is private in this class, except for the interfaces with the framework (need to get some work from somewhere)

68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97

private:

```
// Method to write the histograms in a ROOT file.  
void SaveHistograms();  
  
// Print debug information:  
// _debug      turn on or off the debug information.  
bool _debug;  
  
// Do stupid things for educational purposes (D0 software tutorial):  
// _tutorial    turn on all the stupid things.  
bool _tutorial;  
  
// Buffers for RCP parameters:  
// _emKey       key for accessing the required EM particle chunk;  
// _jetCSEL     jet chunk selector;  
edm::IKey<emid::EMparticleChunk> _emKey;  
jetid::JetChunkSelector _jetCSEL;  
  
// Tool for performing trigger selection.  
D0analysis::D0TriggerSelector *_selectTriggers;  
  
// Histograms:  
// _do_histograms controls whether histograms are filled or not;  
// _histo_saved   controls whether histograms have been written out or not;  
// _hepMgr        pointer to the ROOT method for handling I/O.  
bool _do_histograms;  
bool _histo_saved;  
HepRootFileManager *_hepMgr;
```

D0ChunkAnalyzer.hpp lines 68-97/157 65%

Everything private !!!

Access to trigger info

EM and Jets chunks are rather peculiar, more difficult to access inside the framework (more than 1 algo)


```

99 // List of all the histograms used in the code (ROOT histograms are
100 // used directly instead of using the HepTuple interface).
101
102 // Electron histograms
103 TH1F *_h1numEle;
104 TH2F *_h2emFvsIso;
105 TH1F *_h1EoverP;
106 TH1F *_h1numTkMatch;
107
108 // Muon histograms
109 TH1F *_h1numMu;
110 TH1F *_h1muoPt;
111 TH2F *_h2muGlobLocal;
112
113 // Jet histograms
114 TH1F *_h1numJet;
115 TH1F *_h1jetPt;
116
117 // Missing ET histograms
118 TH1F *_h1SET;
119 TH1F *_h1MET;
120
121 // Tracks histograms
122 TH1F *_h1trkpl;
123 TH1F *_h1trkPhi;
124 TH1F *_h1trkDip;
125 TH1F *_h1trkSigd0;
126 TH1F *_h1trkSMHits;
127 TH1F *_h1trkCFHits;
128 TH1F *_h1trkZ0;
129 TH1F *_h1numVtxMatch;
130 TH2F *_h2trkVtxXY;
131
132 // Simple ROOT tuple.
133 TTree *_treeTracks;
134 Int_t _nTracks;
135 Float_t _fpTTracks[100];
136 Float_t _fetaTracks[100];
137 Float_t _fphiTracks[100];
138

```

00ChunkAnalyze.hpp lines 99-138/157 87%

Create pointers for histograms and a ROOT tuple. The same as in your C++ ROOT macro ?

May be slightly different because
ROOT allows you to code in a language which is not exactly C++ and which allows a lot of mistakes

Always compile your ROOT macros


```

139 // Event counters:
140 // _processed_events    total number of events processed;
141 // _triggered_events    total number of events passing the trigger requirements;
142 // _selected_events     total number of events passing the user selection.
143 int _processed_events;
144 int _triggered_events;
145 int _selected_events;
146
147 // Event tags:
148 // _event_tag           tag used for selecting events.
149 edm::Tag_event_tag;
150
151 // Error log
152 ErrorLog errLog;
153
154 }; // class D0ChunkAnalyze
155
156 } // namespace D0example
157 #endif // INC_D0CHUNKANALYZE
D0ChunkAnalyze.hpp Lines 139-157 (END)

```

Event counting/bookkeeping

Name used for event tagging.

ErrorLog: used to control the space used by logfiles.

All variables and most of the infrastructure are private

And finally the analysis code: D0ChunkAnalyze.cpp

```
1 ///////////////////////////////////////////////////////////////////
2 //
3 //
4 // File: D0ChunkAnalyze.cpp
5 //
6 // Purpose: skeleton for performing data analysis at the D0 chunk level
7 //
8 // Created: 24-JUL-2002 Marco Verzocchi
9 //
10 // History:
11 //
12 ///////////////////////////////////////////////////////////////////
13 //
14 // Dependencies (#includes)
15 //include "analysis_example/D0ChunkAnalyze.hpp"
16
17
18 #include "rcp/RCP.hpp"
19
20 #include "framework/Registry.hpp"
21
22 #include "edm/IDSelector.hpp"
23 #include "edm/LinkPtr.hpp"
24
25 #include "prod_history/HistorySelector.hpp"
26
27 #include "em_evt/EMparticleChunkSelector.hpp"
28 #include "em_evt/EMparticle.hpp"
29 #include "em_evt/EMCluster.hpp"
30 #include "em_evt/EMQualityInfo.hpp"
31
32 #include "chpart_evt/ChargedParticleChunk.hpp"
33 #include "chpart_evt/ChargedParticle.hpp"
34 #include "vertexutil/TrackDCA.hpp"
35 #include "d0track/D0HitMask.hpp"
36
```

Description of your class

More framework, RCP and
EDM related stuff

Event history

EMparticle chunk

Charged particles and vertices

D0ChunkAnalyze.cpp lines 1-36/589 4%

D0 (analysis) software tutorial

10/8/02

10

```

37 #include "muonid/MuonParticleChunk.hpp"
38 #include "muonid/MuonParticle.hpp"
39 #include "muonid/MuonQualityInfo.hpp"
40
41 #include "jet_evt/JetChunk.hpp"
42 #include "jet_evt/JetAlgoInfo.hpp"
43 #include "jet_evt/Jet.hpp"
44
45 #include "missingET/MissingET.hpp"
46 #include "missingET/MissingETChunk.hpp"
47
48 #include "vertex_evt/VertexCollChunk.hpp"
49 #include "vertexutil/Vertex.hpp"
50
51 #include <iostream>
52 #include <iomanip>
53 #include <vector>
54
55 using namespace edm;
56 using namespace std;
57 using namespace fwk;
58 using namespace emid;
59 using namespace muonid;
60 using namespace jetid;
61 using namespace trf;
62 using namespace vertex;
63 using namespace D0analysis;
64
D0ChunkAnalyze.cpp: lines 37-64/589 7%

```

Use namespaces to isolate your code from that of other people

Another list of include statements. For each kind of objects need several of them, at least two, if not more.

Figuring out the right list of required include files is one of the most annoying features of the system

Need proper documentation for each physics object

```
65 // The following lines are here for educational purposes only.
```

```
66
67 #include "ZMutlity/iostream"
68 #ifndef ZMEXCEPTION_H
69 #include "Exceptions/ZMexception.h"
70 #endif // ZMEXCEPTION_H
71 #include "Exceptions/ZMthrow.h"
72
```

```
73 ZM_USING_NAMESPACE (zmex);
74 ZMexStandardDefinition(ZMexception,ZMxD0analysisElvisIsActive);
75 ZMexClassInfo ZMxD0analysisElvisIsActive::_classInfo("ZMxD0analysis", "D0ChunkAnalyze", ZMexERROR );
```

```
76
77 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
78
79 namespace D0example {
```

```
80
81 FWK_REGISTRY_IMPL(D0ChunkAnalyze, "$Name: $")
```

```
82
83 D0ChunkAnalyze::D0ChunkAnalyze(Context* context):
84     Package(context), Process(context), Analyze(context),
85     FileOpen(context), FileClose(context), JobSummary(context) {
86     // Default constructor.
87
```

```
88     cout << endl
89     << "+-----+" << endl
90     << "| D0ChunkAnalyze - package initialization |" << endl
91     << "+-----+" << endl
92     << endl;
93
```

```
D0ChunkAnalyze.cpp lines 65-93/589 11%
```

Used in a few slides to show
how to abort a program

Isolate your code in your own namespace

Line 81: already seen, register this package with the framework
Lines 83.....: constructor, tell the system which interfaces are implemented

```

94 // Access the RCP information.
95 RCP rcp = packageRCP();
96
97 // Print debug information.
98 _debug = rcp.getUntrackedBool("debug",false);
99 cout << "D0ChunkAnalyze: print debug information: " << boolalpha << _debug << endl;
100
101 // Turn on silly things done for the tutorial.
102 _tutorial = rcp.getUntrackedBool("tutorial",false);
103 cout << "D0ChunkAnalyze: do stupid things in the tutorial: " << boolalpha << _tutorial << endl;
104

```

Read the package RCP: D0ChunkAnalyze.rcp

```

105 // Select one of the EM particle chunks.
106 RCP emidAlgoRCP = rcp.getRCP("EMid_Algo");
107 vector<string> emidNestedRCP = rcp.getVString("EMid_SearchRCPs");
108 EMaparticleChunkSelector emAlgoSel(emidAlgoRCP,emidNestedRCP);
109 _emKey = TKey<EMparticleChunk>(emAlgoSel);
110 cout << "D0ChunkAnalyze: select EM particles created by the RCP identifier: " << emidAlgoRCP << endl;
111
112 // Select one of the Jet chunks.
113 string jetType = rcp.getString("JetAlgo_type");
114 std::vector<float> jetValues = rcp.getVFloat("JetAlgo_values");
115 std::vector<std::string> jetNames = rcp.getVString("JetAlgo_names");
116 if ( jetNames.size() > 0 )
117     while ( *jetNames.rbegin() == "" ) jetNames.pop_back();
118 if ( jetValues.size() < jetNames.size() ) {
119     cout << "D0ChunkAnalyze: inconsistent RCP values for the jet selection, disable it" << endl;
120     jetNames.clear();
121     jetValues.clear();
122 }
123 JetAlgoInfo jetAlgoInfo(jetType,jetValues,jetNames);
124 cout << "D0ChunkAnalyze: select jets matching the following criteria: " << endl
125     << jetAlgoInfo << endl;
126 _jetCSel = JetChunkSelector(jetAlgoInfo);

```

D0ChunkAnalyze.cpp Lines 94-126/589 18%

Example of RCP file (first variables are untracked)

There are 2 EMparticle chunks and 8 (or 16 in MC) Jet Chunks, need to build the appropriate barcode for them

Remember the barcode analogy when looking for the right pellet which has the information you're looking for...

- There is a bar code for type (EM, Jets, Muons,...)
- There is a bar code for origin (which EM or Jet Algo)
- The first bar code is the name of the chunk
- To build the second bar code, you need the RCPID
- Numbers are not easy to remember (<8618 1> means Run II k_T jet finder with D=0.4 in D0reco versions 8.04 to 8.13 ??????)
- So pass the relevant quantities from the RCP file, these are compared with the RCP database, and when a match is found the RCPID is generated..... this is your second barcode....


```
// Select the jet finding algorithm
// (Run II 0.5 cone algorithm with preclustering)
// See the RCP files in the jetanalyze package for other possibilities.
string JetAlgo_type = "PreSCilcone"
string JetAlgo_names = ( "towers" "coneSize" "Radius_of_Cone" "Min_Jet_ET")
float JetAlgo_values = ( 0. 0.3 0.5 8. )

// Select the EMparticle chunk:
// simple cone algorithm --> EMReco-scone-id
// cell nearest neighbour algorithm --> EMReco-cnn-id
RCP EMid_Algo = < emreco EMReco-scone-id >
string EMid_SearchRCPs = ("clusterer", "HMReco",)
```

cole/EXAMPLE/analysis_example/rcp/D0ChunkAnalyze.rcp lines 15-27/35 72%

- **First 3 lines:** configuration of the jet finder used to reconstruct jets with the RunII cone algorithm with a radius of 0.5
- **Last 2 lines:** configuration of emreco for the simple cone algorithm

Who is the lucky winner of prize #7 ?

```

127 // Name of the tag assigned to selected events.
128 _event_tag = rcp.getString("Event_tag");
129
130
131 // Initialise event counters.
132 _processed_events = 0;
133 _triggered_events = 0;
134 _selected_events = 0;
135
136 // Trigger selection.
137 _selectTriggers = D0TriggerSelector::Instance();
138
139 // Create histograms.
140 _do_histograms = rcp.getBool("FillHistograms");
141
142 // Initialize ROOT.
143 if ( _do_histograms ) {
144
145     _histo_saved = false;
146
147     // Open the file which will contain the histograms.
148     string histoFile = rcp.getString("HistogramFile");
149     cout << "D0ChunkAnalyze: histograms will be written to: " << histoFile << endl;
150     _hepMgr = new HepRootFileManager(histoFile.c_str(),"");
151
152     // Book the user histograms.
153
154     // Histograms for EM particles.
155     _h1numEle = new TH1F("numEle","Number of EM candidate (simple cone algorithm)",21,-0.5,20.5);
156     _h2emFvsIso = new TH2F("emFvsIso","EM fraction versus Isolation (id=10,+/-11)",60,-0.1,0.2,60,0.8,1.1);
157     _h1numTkMatch = new TH1F("numTkMatch","Number of tracks matched to the EM candidate",11,-0.5,10.5);
158     _h1EoverP = new TH1F("EoverP","E/p ratio for selected EM candidates",100,0..3.);
159

```

Event counters

Pointer to the package which
does select events based on
their triggers

We're still in the **constructor**, reading RCP parameters,
creating objects needed to extract informations from the data,
creating a bunch of histograms

D0ChunkAnalyze.cpp lines 127-159/589 23%

```

160 // Histograms for muons.
161 _h1numMuo = new TH1F("numMuo", "Number of muon candidates", 11, -0.5, 10.5);
162 _h1muoPt = new TH1F("muoPt", "Muon candidates transverse momentum distribution", 50, 0., 100.);
163 _h2muGlobLocal = new TH2F("muGL", "Central tracker vs local muon p measurement", 50, -50., 50., 50, -50., 50.);
164
165 // Histograms for jets.
166 _h1numJet = new TH1F("numJet", "Number of jets (JCCB algorithm)", 11, -0.5, 10.5);
167 _h1jetPt = new TH1F("jetPt", "Jets (JCCB) transverse energy distribution", 50, 0., 100.);
168
169 // Histograms for missing Et.
170 _h1SET = new TH1F("SET", "Total scalar Et distribution", 100, 0., 300.);
171 _h1MET = new TH1F("MET", "Missing transverse momentum distribution", 50, 0., 100.);
172
173 // Histograms for tracks.
174 _h1trkpt = new TH1F("trkPt", "Transverse momentum of tracks", 50, 0., 50.);
175 _h1trkPhi = new TH1F("trkPhi", "Azimuthal angle of tracks", 50, 0., 6.3);
176 _h1trkDip = new TH1F("trkDip", "Dip angle of tracks", 100, -20., 20.);
177 _h1trkSigd0 = new TH1F("trkSigd0", "Impact parameter significance", 100, -20., 20.);
178 _h1trkSMThits = new TH1F("trkSMThits", "Number of hits in the SMT", 21, -0.5, 20.5);
179 _h1trkCFThits = new TH1F("trkCFThits", "Number of hits in the CFT", 21, -0.5, 20.5);
180 _h1trkZ0 = new TH1F("trkZ0", "Z coordinate of DCA point (relative to best vertex)", 100, -30., 30.);
181 _h1numVtxMatch = new TH1F("trkVtxMatch", "Number of vertices matched to tracks", 21, -0.5, 20.5);
182 _h2trkVtxXY = new TH2F("trkVtxXY", "XY distribution of primary vertices", 50, -1.1, 50., -1.1, 1.);
183
184 // Create a simple ROOT tuple.
185 _ttreeTracks = new TTree("treeTracks", "A very simple ROOT tuple containing tracks");
186 _ttreeTracks->Branch("nTrk", &nTrks, "nTrk/I");
187 _ttreeTracks->Branch("pT", &fpTTracks, "pT[nTrk]/F");
188 _ttreeTracks->Branch("phi", &fphiTracks, "phi[nTrk]/F");
189 _ttreeTracks->Branch("eta", &fetaTracks, "eta[nTrk]/F");
190
191 }
192
193 // Finish initialization.
194 cout << endl;
195 << "+-----+" << endl;
196 << "| D0ChunkAnalyze - initialization finished |" << endl;
197 << "+-----+" << endl;
198 << endl;
199
200 }
D0ChunkAnalyze.cpp lines 160-200/589 32%

```

After all those
histograms create a
simple ROOT tuple

```

202 Result D0ChunkAnalyze::processEvent(Event &event) {
203 // Process one event.
204
205 // Increase the counter for the number of events processed.
206 ++_processed_events;
207
208 // Check whether the event satisfies the user trigger requirements.
209 if ( _selectTriggers->triggerFired(event) ) ++_triggered_events;
210
211 // -----
212 // The following lines are here only for educative purposes. Remove them from
213 // the code if you plan on doing any serious use of D0ChunkAnalyze.
214 // -----
215 if ( _tutorial ) {
216
217 // Print out a stupid message using the error logger.
218 errlog.setModule("D0example::D0ChunkAnalyze");
219 errlog(ELinfo,"processEvent") << "Elvis has just left the building" << endl;
220
221 } // End of lines inserted in the code for educative purposes.
222
223 return Result::success;
224
225 }
226

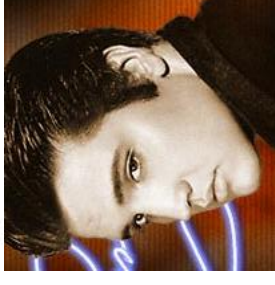
```

D0ChunkAnalyze.cpp lines 202-226/589 36%

processEvent: can add information to the event (not done here)

here I'm just counting the # of the events which fire the trigger
(see part 4 for how the trigger list is specified) at line 209

Do stupid things:
it's a tutorial !!!!!



The proper way of writing "Elvis has just left the building"*

- ♦ There is no point in writing a stupid line of output for every event
- ♦ You're probably not annoyed because you test with just 100 events
- ♦ Multiply by 200 packages, and by 25 (typical farm file contains 2500 events)
 - Heidi & Mike get annoyed
- ♦ The same thing happens at L3: each process on each node will see few Hz of data. There is no such a thing as a file size... You go on and on an on and on.....
 - ♦ until you crash the farm node because somebody has to tell the world that "Elvis has just left the building"
 - Andy, Doug, Gordon, Gustaaf, Mike and Run get upset

* F. Zappa, *Broadway the hard way*, 1989

Use the error logger instead.....

FORBIDDEN

~~`cout << "Elvis has just left the building" << endl;`~~

INTERDIT

VERBOTEN

```
#include "ErrorLogger/ErrorLog.h"
```

```
ErrorLog errlog;
```

```
errlog.setModule("D0example::D0ChunkAnalyze")
```

```
errlog(ELinfo, "processEvent") <<
```

```
    "Elvis has just left the building" << endlmsg;
```

This is the right way
of printing out
messages from your
code

-i- means information not error !!!

In D0ChunkAnalyze_x.log you will see only a few times:

```
%ERLOG-i processEvent: Elvis has just left the building
```

```
    D0example::D0ChunkAnalyze 7-Oct-2002 22:21:11
```

```
    D0ChunkAnalyze:analyze Run number: 160196, Event Number: 47422895
```

plus one additional line at the end which will tell you
how often this message occurred:

```
9 processEvent
```

```
    -i D0example::D0Chu
```

```
    227* 227
```


What if the message is more serious ("Elvis is dead") ?

<i>Severity object</i>	<i>Symbol</i>	<i>Full name</i>	<i>Intention</i>
<i>ELzeroSeverity</i>	--	--	
<i>ELincidental</i>	<i>flash this on a screen</i>
<i>ELsuccess</i>	-!	SUCCESS	<i>report reaching a milestone</i>
<i>ELinfo</i>	-i	INFO	<i>information</i>
<i>ELwarning</i>	-w	WARNING	<i>warning</i>
<i>ELwarning2</i>	-W	WARNING!	<i>more serious warning</i>
<i>ELerror</i>	-e	ERROR	<i>error detected</i>
<i>ELerror2</i>	-E	ERROR!	<i>more serious error</i>
<i>ELnextEvent</i>	-n	NEXT	<i>advise to skip to next event</i>
<i>ELunspecified</i>	??	??	<i>severity was not specified</i>
<i>ELsevere</i>	-s	SEVERE	<i>future results are suspect</i>
<i>ELsevere2</i>	-S	SEVERE!	<i>more severe</i>
<i>ELabort</i>	-A	ABORT!	<i>suggest aborting</i>
<i>ELfatal</i>	-F	FATAL!	<i>strongly suggest aborting!</i>
<i>ElhighestSeverity</i>	!!	!!	

<http://www-d0.fnal.gov/d0dist/dist/packages/ErrorLogger/devel/doc/html/physicist.html>

Doing all the work in analyzeEvent

```
227 Result D0ChunkAnalyze::analyzeEvent(const Event &event) {  
228 // Fill histograms.
```

```
229  
230 // Run and event number.  
231 int eventNum = static_cast<int> (event.collisionID().eventNumber());  
232 int runNum = static_cast<int> (event.collisionID().runNumber());  
233
```

Run and event #
are in collisionID

```
234 // -----  
235 // The following lines are here only for educative purposes. Remove them from  
236 // the code if you plan on doing any serious use of D0ChunkAnalyze.  
237 // -----  
238 if ( _tutorial ) {
```

```
239  
240 // Print another stupid message, if the event number is even.  
241 if ( (eventNum%2)==0 ) {  
242     errlog.setModule("D0example::D0ChunkAnalyze");  
243     errlog(ELInfo,"analyzeEvent") << "Elvis is dead" << endl;   
244 }  
245
```

```
246 // Throw an exception if this is MC.  
247 bool isthisMC = HistorySelector::is_monte_carlo(event);  
248 if ( !isthisMC ) {
```

data or MC

```
249     ostreamstream msg;  
250     msg << "You attempted to look at the MC with the D0ChunkAnalyze package\n"  
251     << "I know that you are really looking for Elvis, but no not at D0\n"  
252     << "and you should not use the MC to convince us that Elvis is alive\n"  
253     << ends;
```

```
254     ZMthrow(ZMxD0analysisElvisIsAlive(msg.str()));  
255 }  
256
```

Throwing exceptions

```
257 if ( _debug ) {  
258     cout << "Inside D0ChunkAnalyze::analyzeEvent triggers ";  
259     _selectTriggers->PrintFiredTriggers(event,cout);  
260     cout << endl;  
261 }  
262
```

a way to force
the program to stop

```
263 } // End of lines inserted in the code for educative purposes.  
264
```

D0ChunkAnalyze.cpp lines 227-264/589 43%

MORE STUPID THINGS

Stupid things are sometimes necessary.....

Finally it's time for data access !!!!! The EMparticleChunk

```
265 // -----
266 // Access the EM particle chunk
267 // -----
268 THandle<EMparticleChunk> emChunk=_emKey.find(event);
269
270 // Check whether a non empty EM particle chunk was found
271 int numEle = 0;
272 if ( emChunk.isValid() ) {
273
274     vector<EMparticle>::const_iterator empitr;
275     const vector<EMparticle>* pvector = emChunk->getParticles();
276
277     // Number of electrons
278     numEle = pvector->size();
279
280     // Loop on the electron/photon candidates.
281     for ( empitr=pvector->begin(); empitr!=pvector->end(); ++empitr) {
282
283         int emType = abs(empitr->typeID());
284         // EM particle type
285
286         // Consider only EM particles of type 10 and 11.
287         if ( emType==10 || emType==11 ) {
288             const EMQualityInfo* pqual = empitr->qualInfo(); // pointer to the quality information
289             const EMCluster* pclus = pqual->emclusptr(); // pointer to the cluster
290         }
291     }
292 }
```

D0ChunkAnalyze.cpp Lines 265-288/589 47%

Line 268: read the bar code and find the right chunk


Line 272: check that there is something in the chunk

Line 274: get something to scan all the EM objects in the chunk

Line 275: fill a container with all those EM objects in the chunk

The EMparticleChunk

```
265 // -----
266 // Access the EM particle chunk
267 // -----
268 THandle<EMparticleChunk> emChunk=_emKey.find(event);
269
270 // Check whether a non empty EM particle chunk was found
271 int numEle = 0;
272 if ( emChunk.isValid() ) {
273
274     vector<EMparticle>::const_iterator emptr;
275     const vector<EMparticle>* pvector = emChunk->getParticles();
276
277     // Number of electrons
278     numEle = pvector->size();
279
280     // Loop on the electron/photon candidates.
281     for ( emptr=pvector->begin(); emptr!=pvector->end(); ++emptr) {
282
283         int emType = abs(emptr->TypeID()); // EM particle type
284
285         // Consider only EM particles of type 10 and 11.
286         if ( emType==10 || emType==11 ) {
287             const EMQualityInfo* pqual = emptr->qualInfo(); // pointer to the quality information
288             const EMCluster* pclus = pqual->emclusptr(); // pointer to the cluster
289         }
290     }
291 }
```



D0ChunkAnalyze.cpp lines 265-288/589 47%

Line 281: loop over all those EM objects in the container

This sequence of 5 operations is always the same for most D0 physics object chunks..... The difference is really in the contents of each object (EM, jets,)

Link indices and link pointers:

purpose) an electron may have one or more tracks associated to it: for each possible association (**link index**) need a pointer (**link pointer**) to the track object it also has a set of calorimeter cells, need pointers to them as wellm and to the preshower....

```
296 // Example of using link indices: retrieve the list of tracks
297 // matched to the EM candidate
298 int numTkMatch = emptr->chpindices().size(); // number of tracks matched to the EM particle
299 if ( _do_histograms ) _h1numTkMatch->Fill(numTkMatch);
300
301 // Loop on the tracks
302 if ( numTkMatch ) {
303     for (int j=0; j!=numTkMatch; ++j ) {
304         LinkPtr<ChargedParticleChunk,ChargedParticle> trkptr(emptr->chpindices()[j]);
305
306         // Check whether this is a valid pointer to a track
307         if ( trkptr.isValid() ) {
00ChunkAnalyze.cpp lines 296-307/589 51%
```

Line 298: find # of associations

Line 304: for each association get the pointer to the track

Line 307: always check (as for chunks) that the framework has given you a valid pointer (isValid()**)**

The other chunks: now it's just repetition.....muons

```
326 // -----  
327 // Access the muon particle chunk  
328 // -----  
329 TKey<MuonParticleChunk> mupKey;  
330 THandle<MuonParticleChunk> muChunk=mupKey.find(event);  
331  
332 // Check whether a non empty muon particle chunk was found  
333 int numMu0 = 0;  
334 if ( muChunk.isValid() ) {  
335  
336     vector<MuonParticle>::const_iterator mupptr;  
337     const vector<MuonParticle>* pvector=muChunk->getParticles();  
338  
339     // Number of muons  
340     numMu0 = pvector->size();  
341  
342     // Loop on the muon candidates.  
343     for ( mupptr=pvector->begin(); mupptr!=pvector->end(); ++mupptr ) {  
344  
345         const muonid::MuonQualityInfo *muonquality = mupptr->qualInfo(); // pointer to the quality information  
346  
347     }  
348 }  
349  
350 }  
351  
352 }  
353  
354 }  
355  
356 }  
357  
358 }  
359  
360 }  
361  
362 }  
363  
364 }  
365  
366 }  
367  
368 }  
369  
370 }  
371  
372 }  
373  
374 }  
375  
376 }  
377  
378 }  
379  
380 }  
381  
382 }  
383  
384 }  
385  
386 }  
387  
388 }  
389  
390 }  
391  
392 }  
393  
394 }  
395  
396 }  
397  
398 }  
399  
400 }  
401  
402 }  
403  
404 }  
405  
406 }  
407  
408 }  
409  
410 }  
411  
412 }  
413  
414 }  
415  
416 }  
417  
418 }  
419  
420 }  
421  
422 }  
423  
424 }  
425  
426 }  
427  
428 }  
429  
430 }  
431  
432 }  
433  
434 }  
435  
436 }  
437  
438 }  
439  
440 }  
441  
442 }  
443  
444 }  
445  
446 }  
447  
448 }  
449  
450 }  
451  
452 }  
453  
454 }  
455  
456 }  
457  
458 }  
459  
460 }  
461  
462 }  
463  
464 }  
465  
466 }  
467  
468 }  
469  
470 }  
471  
472 }  
473  
474 }  
475  
476 }  
477  
478 }  
479  
480 }  
481  
482 }  
483  
484 }  
485  
486 }  
487  
488 }  
489  
490 }  
491  
492 }  
493  
494 }  
495  
496 }  
497  
498 }  
499  
500 }  
501  
502 }  
503  
504 }  
505  
506 }  
507  
508 }  
509  
510 }  
511  
512 }  
513  
514 }  
515  
516 }  
517  
518 }  
519  
520 }  
521  
522 }  
523  
524 }  
525  
526 }  
527  
528 }  
529  
530 }  
531  
532 }  
533  
534 }  
535  
536 }  
537  
538 }  
539  
540 }  
541  
542 }  
543  
544 }  
545  
546 }  
547  
548 }  
549  
550 }  
551  
552 }  
553  
554 }  
555  
556 }  
557  
558 }  
559  
560 }  
561  
562 }  
563  
564 }  
565  
566 }  
567  
568 }  
569  
570 }  
571  
572 }  
573  
574 }  
575  
576 }  
577  
578 }  
579  
580 }  
581  
582 }  
583  
584 }  
585  
586 }  
587  
588 }  
589  
590 }  
591  
592 }  
593  
594 }  
595  
596 }  
597  
598 }  
599  
600 }  
601  
602 }  
603  
604 }  
605  
606 }  
607  
608 }  
609  
610 }  
611  
612 }  
613  
614 }  
615  
616 }  
617  
618 }  
619  
620 }  
621  
622 }  
623  
624 }  
625  
626 }  
627  
628 }  
629  
630 }  
631  
632 }  
633  
634 }  
635  
636 }  
637  
638 }  
639  
640 }  
641  
642 }  
643  
644 }  
645  
646 }  
647  
648 }  
649  
650 }  
651  
652 }  
653  
654 }  
655  
656 }  
657  
658 }  
659  
660 }  
661  
662 }  
663  
664 }  
665  
666 }  
667  
668 }  
669  
670 }  
671  
672 }  
673  
674 }  
675  
676 }  
677  
678 }  
679  
680 }  
681  
682 }  
683  
684 }  
685  
686 }  
687  
688 }  
689  
690 }  
691  
692 }  
693  
694 }  
695  
696 }  
697  
698 }  
699  
700 }  
701  
702 }  
703  
704 }  
705  
706 }  
707  
708 }  
709  
710 }  
711  
712 }  
713  
714 }  
715  
716 }  
717  
718 }  
719  
720 }  
721  
722 }  
723  
724 }  
725  
726 }  
727  
728 }  
729  
730 }  
731  
732 }  
733  
734 }  
735  
736 }  
737  
738 }  
739  
740 }  
741  
742 }  
743  
744 }  
745  
746 }  
747  
748 }  
749  
750 }  
751  
752 }  
753  
754 }  
755  
756 }  
757  
758 }  
759  
760 }  
761  
762 }  
763  
764 }  
765  
766 }  
767  
768 }  
769  
770 }  
771  
772 }  
773  
774 }  
775  
776 }  
777  
778 }  
779  
780 }  
781  
782 }  
783  
784 }  
785  
786 }  
787  
788 }  
789  
790 }  
791  
792 }  
793  
794 }  
795  
796 }  
797  
798 }  
799  
800 }  
801  
802 }  
803  
804 }  
805  
806 }  
807  
808 }  
809  
810 }  
811  
812 }  
813  
814 }  
815  
816 }  
817  
818 }  
819  
820 }  
821  
822 }  
823  
824 }  
825  
826 }  
827  
828 }  
829  
830 }  
831  
832 }  
833  
834 }  
835  
836 }  
837  
838 }  
839  
840 }  
841  
842 }  
843  
844 }  
845  
846 }  
847  
848 }  
849  
850 }  
851  
852 }  
853  
854 }  
855  
856 }  
857  
858 }  
859  
860 }  
861  
862 }  
863  
864 }  
865  
866 }  
867  
868 }  
869  
870 }  
871  
872 }  
873  
874 }  
875  
876 }  
877  
878 }  
879  
880 }  
881  
882 }  
883  
884 }  
885  
886 }  
887  
888 }  
889  
890 }  
891  
892 }  
893  
894 }  
895  
896 }  
897  
898 }  
899  
900 }  
901  
902 }  
903  
904 }  
905  
906 }  
907  
908 }  
909  
910 }  
911  
912 }  
913  
914 }  
915  
916 }  
917  
918 }  
919  
920 }  
921  
922 }  
923  
924 }  
925  
926 }  
927  
928 }  
929  
930 }  
931  
932 }  
933  
934 }  
935  
936 }  
937  
938 }  
939  
940 }  
941  
942 }  
943  
944 }  
945  
946 }  
947  
948 }  
949  
950 }  
951  
952 }  
953  
954 }  
955  
956 }  
957  
958 }  
959  
960 }  
961  
962 }  
963  
964 }  
965  
966 }  
967  
968 }  
969  
970 }  
971  
972 }  
973  
974 }  
975  
976 }  
977  
978 }  
979  
980 }  
981  
982 }  
983  
984 }  
985  
986 }  
987  
988 }  
989  
990 }  
991  
992 }  
993  
994 }  
995  
996 }  
997  
998 }  
999  
1000 }
```

Lines 329,330: create a bar code and search for the MuonParticle chunk in the event (was 1 line for electrons, but the work was done beforehand)

Line 334: line 272 for EM objects (identical)

Line 336: line 274 for EM objects

Line 337: line 275 for EM objects

Line 343: line 281 for EM objects

By learning 5 or 6 lines of code, you can access any D0 physics object

```

342 // Loop on the muon candidates.
343 for ( mupptr=pvector->begin(); mupptr!=pvector->end(); ++mupptr ) {
344
345     const muonid::MuonQualityInfo *muonquality = mupptr->qualInfo(); // pointer to the quality information
346
347     // Decode the muon quality words
348     int nSegMu = muonquality->nseg();
349     int nHitMu = muonquality->nhit();
350     float chiSqMu = muonquality->chisqloc();
351     int nMSchHit[5];
352     for ( int ihit=0; ihit!=5; ++ihit )
353         nMSchHit[ihit] = 0;
354     int ii = 0;
355     while ( nHitMu>0 ) {
356         nMSchHit[ii] = nHitMu%10;
357         nHitMu /= 10;
358         ++ii;
359     } // Decoding muon quality information
360     int nMHit = nMSchHit[0];
361     int nBCWHit = nMSchHit[1] + nMSchHit[2]*10;
362     int nASHit = nMSchHit[3];
363     int nBCSHit = nMSchHit[4];
364
365     // Apply the tight muon selection cuts
366     if ( nMSchHit[0]>=2 && nBCWHit>=2 &&
367         nMSchHit[3]>=1 && nMSchHit[4]>=1 && chiSqMu>0. ) {
368
369         float globalpT = mupptr->pT(); // best measurement of the muon pT
370         if ( !_do_histograms ) _h1muoPt->Fill(globalpT);
371         globalpT = globalpT*mupptr->charge(); // sign the muon pT with the charge of the muon
372
373         // For muons matched to central tracks compare the two possible muon momentum measurements
374         if ( nSegMu>0 ) {
375             float localpT = muonquality->ptloc(); // local measurement of the muon pT
376             if ( !_do_histograms ) _h2muGlobLocal->Fill(localpT,globalpT);
377
378             // Muons matched to central tracks
379             // Tight muon selection
380             // Loop on muon candidates
381         }
382     }

```

D0Chunk0analyze.cpp lines 342-382/589 65%

Decode μ quality information

Tight muon requirement

still an electron and a muon
share some properties and others
instead are completely different

Shared properties (by most D0 physics objects):

pX(), pY(), pZ(), pT(), E(), eta(), phi(), theta(), p(), type()

completely different:

electron: shower shape, isolation, EM fraction, H matrix X^2 ,....

muon: energy deposited in calorimeter, scintillator timing,....

Complication of doing analysis with D0 physics objects due to the **lack of documentation on the content of the various classes !!!**

More on this in part 4

The Jet Chunk

```




389 // -----
390 // Access the jet particle chunk
391 // -----
392  const TKey<JetChunk> jetKey(_jetCSel);
393 THandle<JetChunk> _jhandle = jetKey.find(event);
394
395 // Check whether a non empty jet particle chunk was found
396 int numJet = 0;
397 if ( _jhandle.isValid() ) {
398
399  JetChunk* jetChunk=_jhandle.ptr();
400  vector<Jet>::const_iterator jetptr;
401  const vector<Jet>* pvector = jetChunk->getParticles();
402
403 // Number of jets
404 numJet = pvector->size();
405
406 // Loop on all the jets and copy the pointers into a list
407 list<const Jet*> listJet;
408  for ( jetptr=pvector->begin(); jetptr!=pvector->end(); ++jetptr )
409     listJet.push_back(*jetptr);
410
411 if ( numJet ) {
412     // Sort the jets in order of decreasing Pt
413     listJet.sort(JetPtOrder());
414
415     // Apply some simple quality cuts on the leading jet
416     list<const Jet*>::iterator iJet;
417     iJet = listJet.begin();
418     int jetN90 = (*iJet)->n90();
419
420     int jetEMfrac = (*iJet)->emEfraction();
421     if ( jetN90>0 && jetEMfrac<0.95 ) {
422         float jetpT = (*iJet)->pT();
423         if ( _do_histograms ) _h1JetPt->Fill(jetpT);
424     } // Jet quality requirements
425
426     } // Jet(s) found
427
428     } // Valid jet chunk
429
430     if ( _do_histograms ) _h1numJet->Fill(static_cast<float>(numJet));

```

D0ChunkFinalize.cpp lines 389-430/589-732

This code is identical to
the one of muons.....

The missing E_T chunk

```
432 // -----  
433 // Access the missing Et chunk  
434 // -----  
435  const TKey<MissingETChunk> metKey;  
436 THandle<MissingETChunk> metchunk = metKey.find(event);  
437  
438 // Check whether a non empty missing ET chunk was found  
439  if ( metchunk.isValid() ) {  
440  
441 // Fill histograms  
442  MissingET* met = metchunk.ptr()->getMissingET(); // pointer to the missing ET  
443 float scalarET = met->getScalarET(); // scalar ET of the event  
444 float missingET = met->getMET(); // missing transverse momentum  
445 if ( _do_histograms ) h1SET->Fill(scalarET);  
446 if ( _do_histograms ) h1MET->Fill(missingET);  
447  
448 } // Valid missing ET chunk
```

D0ChunkAnalyze.cpp lines 432-448/589 76%

An even easier object: other chunks contain vector of objects. The missing E_T chunk contains just one object per event, access even easier.

For now the **missing E_T calculation in D0reco is not too reliable**. Use the cell (CalDataChunk) information during the analysis to get a better estimate of the missing E_T

The ChargedParticleChunk (tracks)

```
450 // -----
451 // Access the track chunk
452 // -----
453 TKey<ChargedParticleChunk> chpartKey;
454 THandle<ChargedParticleChunk> chpartChunk = chpartKey.find(event);
455
456 // Counter for the number of tracks in the ROOT tuple.
457 _nTracks = 0;
458
459 // Check whether a non empty track chunk is found
460 int numTrk = 0;
461 if ( chpartChunk.isValid() ) {
462
463     vector<ChargedParticle>::const_iterator tkptr;
464     const vector<ChargedParticle>* pvector = chpartChunk->getParticles();
465
466     // Number of tracks
467     numTrk = pvector->size();
468
469     // Loop on the tracks
470     for ( tkptr=pvector->begin(); tkptr!=pvector->end(); ++tkptr ) {
471
```

D0ChunkAnalyze.cpp Lines 450-471/589 80%

This is **exactly once more the same access pattern**,..... The coloured arrows on the left help identifying the same lines of code shown previously.

```

470 for ( tkptr=pvector->begin(); tkptr!=pvector->end(); ++tkptr ) {
471
472     // Get pointers to the ETrack and TrackDCA
473     // classes for the current track
474     const ETrack etrack = tkptr->get_etrack();           // track parameters [relative to 0,0,0 ?]
475     const TrackDCA dcaDist = tkptr->get_trackDCA();      // impact parameter relative to best vertex
476     const Vertex* vtxptr = tkptr->get_vtxptr();         // pointer to the vertex
477     d0track::D0HitMask hitmask = error(SurfDCA::IRSIGNED,SurfDCA::IRSIGNED); // error on impact parameter
478     if ( _do_histograms ) {
479         _h1trkDip->Fill(tanIam);
480         if ( sd0!=0. ) _h1trkSigd0->Fill(d0/sd0);
481     }
482
483     int numSMTHits = hitmask.num_smt_layers();          // number of hits in the SMT
484     int numCFTHits = hitmask.num_cft_barrels();        // number of hits in the CFT
485     if ( _do_histograms ) {
486         _h1trkSMTHits->Fill(numSMTHits);
487         _h1trkCFTHits->Fill(numCFTHits);
488     }
489
490     float z0 = dcaDist.dca(TrackDCA::IZ); // z0 of the primary vertex (relative to best vertex)
491     if ( _do_histograms ) _h1trkZ0->Fill(z0);
492
493     // Example of using link indices: retrieve the list of primary vertices to which
494     // this track can be attached
495     int numVtxMatch = tkptr->pvtIndices().size(); // number of primary vertices matched to the track
496     if ( _do_histograms ) _h1numVtxMatch->Fill(numVtxMatch);
497
498     // Loop on the vertices
499     if ( numVtxMatch ) {
500         for ( int j=0; j!=numVtxMatch; ++j ) {
501             LinkPtr<VertexCollChunk,Vertex> vtxptr(tkptr->pvtIndices()[j]);
502
503             // Check whether this is a valid pointer to a primary vertex
504             if ( !vtxptr.isValid() ) {
505                 float xvtx = vtxptr->get_vertexpos(0); // X position of the primary vertex
506                 float yvtx = vtxptr->get_vertexpos(1); // Y position of the primary vertex
507                 if ( _do_histograms ) _h2trkVtxXY->Fill(xvtx,yvtx);
508             }
509         }
510     }
511
512     // Valid vertex pointer
513     } // Loop on vertices
514
515     // Vertices matched to the track
516
517     // Loop on tracks
518
519     } // valid track chunk
520
521     // Fill the ROOT tuple
522     if ( _nTracks>0 ) _ttreeTracks->Fill();
523
524     return Result::success;
525 }
526
527 D0ChunkAnalyze.cpp lines 470-524/589-892

```

Use names, not indices

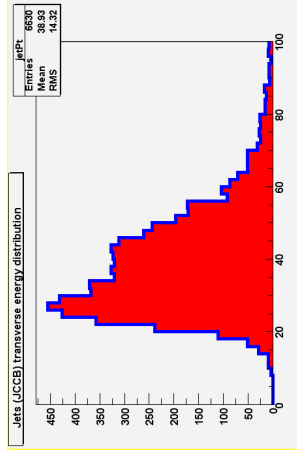
LinkIndices and
LinkPointers at work

Fill the ROOT tuple
Return Result::success;

All the difficulty of writing analysis code inside the D0 framework boils down to:

- taking D0ChunkAnalyze and customizing it
- for 5 D0 physics object chunks there is already a description of the access method
- for the others it is missing
- once you have the pointer to an EMparticle (for example), you would need a list of all the other quantities which are available
- this is really the only difficulty

HOMEWORK: add at least one histogram in the code and then print it....



**End of part 3
reconvene at 11:10
(sharp)**